

- Restoring said state and data of said host OS by un-reserving said reserved memories used by said host OS upon exiting said guest OS;

- Exiting said guest OS and Returning back to said host OS.

~~A system allowing mobile device to run multiple operating systems and preserve the state and data of the original operating system, comprising the steps of:~~

~~-Preparing a guest OS image;~~

~~-Packaging the image into a special host OS application or file;~~

~~-Starting a special boot loader for the host OS to read, unpack, load and start the guest OS image from the special host OS application;~~

~~-Preserving the current state and data of the host OS and starting the guest OS;~~

~~-Exiting from the guest OS, running exiting code to restore the system state and data to the original host OS and then return back to the host OS.~~

2. (Currently Amended) The method of claim 1, wherein, Running said native application from said host OS to reserve memories used by said host OS as unchangeable or invisible from said guest OS to preserve the current state and data of said host OS; further comprising steps:

Said native application manipulating Memory Management Unit (MMU) of the device to manipulating memory protection attributes for memory pages in said host OS;

Reserving memories of said host OS by changing said memory protection attributes for all memories (data and code) used by said host OS to read-only or un-accessible;

~~The system of claim 1, wherein the mobile device is a PDA or Win CE device, cellular phone or other mobile devices.~~

3. (Currently Amended) The method of claim 1, wherein Running said native application from said host OS to reserve memories used by said host OS as

unchangeable or invisible from said guest OS to preserve the current state and data of said host OS; further comprising steps of:

Moving all memories (data and code) of said host OS to top of physical memory of the device;

Generating a faked reduced-memory size to be used by said guest OS;

Preserving said memories (data and code) of said host OS by guest OS uses said faked reduced physical memory size on running and never touching memories above the said faked reduced memory size

~~The system of claim 1, wherein the original host OS cannot share its memory with any other OS concurrently.~~

4. (Currently Amended) The method of claim 3, wherein memories of said mobile device are divided into multiple memory banks, Generating a faked reduced-memory size to be used by said guest OS further comprising:

Generating said faked reduced memory size by marking or disabling one of more said memory banks to make them no longer available.

~~The system of claim 1, wherein the original host OS can only start applications packed with some special formats.~~

5. (Currently Amended) The method of claim 3, Generating a faked reduced-memory size to be used by said guest OS further comprising:

Generating said faked reduced memory size by changing memory detection module of said guest OS to use said faked maximum memory instead of all available memories of said mobile device.

~~The system of claim 1, further comprising a step of preserving the state and data of the guest OS when user wants to temporarily switch away from the guest OS.~~

6. (Currently Amended) The method of claim 3, Generating a faked reduced-memory size to be used by said guest OS further comprising:

Generating said faked reduced memory size by modifying special system registers or IO ports to report said reduced memory size as total available memories to said guest OS.

~~The system of claim 1, wherein the boot loader is either a standalone host OS application or built into the guest OS image wrapper application.~~

7. (Currently Amended) The method of claim 1, wherein Running said native application from said host OS to reserve memories used by said host OS as unchangeable or invisible from said guest OS to preserve the current state and data of said host OS; further comprising steps:

Passing all memories (data and code) of said host OS to a memory device driver being loaded by said guest OS during its initialization;

Said memory device driver of said guest OS claiming all said memories of said host OS and keeping them from being modified by any other part of said guest OS until exit.

~~The system of claim 1, wherein the guest OS image is compressed.~~

8. (Currently Amended) The method of claim 1, wherein Running said native application from said host OS to reserve memories used by said host OS as unchangeable or invisible from said guest OS to preserve the current state and data of said host OS; further comprising steps of:

Saving the whole OS memory image (data or code) of the said host OS into an external memory device;

Restoring said host OS from said OS memory image (data or code) from said external memory device upon retuning from said guest OS.

~~The system of claim 1, wherein the guest OS will use up all memories available.~~

9. (Currently Amended) The method of claim 1, further comprising a step of:
Saving state and data of said guest OS before switching back to said host
OS;
Restoring said state and data of said guest OS when re-entering said
guest OS to continue the previous work in said guest OS;

~~The system of claim 1, further comprising a step of saving the state and data of the guest OS, so users will not lose its work in the guest OS and can return back after switching to original host OS.~~

10. (Currently Amended) The method of claim 1, to start a second guest OS from within said guest OS further comprising steps of:
The said guest OS acting as a host OS to the second guest OS;
The said second guest OS acting as a guest OS;
Repeating steps in Claim 1 to start the second guest OS from with the
said guest OS.

~~The system of claim 1, further comprising starting additional guest OS from either host OS or within the guest OS using the same procedure.~~

11. (Currently Amended) The method of claim 1, wherein running said native application from said host OS to reserve memories used by said host OS as
unchangeable or invisible from said guest OS to preserve the current state and
data of said host OS; further comprising steps of:
Moving memory blocks used by host OS but also needed by said guest
OS to a free memory location in the device and reserve said free memory blocks;
Wherein Restoring said state and data of said host OS by un-reserving
said reserved memories used by said host OS upon exiting said guest OS,
further comprising steps of:

Restoring said memory blocks used by host OS from said free memory blocks;

~~The system of claim 1, wherein the guest OS image is stored in a memory card as a regular file while host OS can access regular files in memory card.~~

12. (Currently Amended) The method of claim 1, wherein said guest OS image is stored in a memory card and can be started immediately when said memory card is inserted into said mobile device.

~~The system of claim 1, wherein the guest OS image is stored in a memory card and can be started immediately when the card is inserted into the device.~~

13. (Currently Amended) The method of claim 1, wherein said native application in host OS reserves only dynamic runtime memory of said host OS and do not reserve memories that can be restored from flash memory in the said mobile device.

~~The system of claim 1, wherein the boot loader only preserves the modified or used memory of host OS and do not care about free or unused memory blocks in the system.~~

14. (Currently Amended) The method of claim 1, wherein Input Output (IO) states, registers of mobile device within said host OS are preserved into memories and later restored from memories.

~~The system of claim 1, wherein IO states or registers of peripherals are saved in memories and preserved the same as other memories and then after memories are restored, restore IO states or registers from the memories.~~

15. (Currently Amended) The method of claim 1, wherein said host OS or said guest OS can be Palm OS, Windows CE, Symbian, Embedded Linux, mobile operating systems.

~~The system of claim 1, wherein the host OS or guest OS can be Palm OS, Windows CE, Symbian, Embedded Linux or other mobile operating systems.~~

16. (Currently Amended) A method of packaging an image of guest OS inside a native application of a host OS to allow in-place execution of the said guest OS image in order to reduce memory usage comprising the steps of:

Compiling an image of guest OS into multiple code segments;

Appending each said code segment with a corresponding jump table containing multiple jump instructions to allow inter code-segment invocation;

Pre-linking said each inter code-segment invocation in each said code segments of guest OS to a said jump instruction in the said jump table corresponding to each said code segment;

Preparing a native application for said host OS with a startup code and multiple data chunks each large enough to contain each of said multiple code segments of guest OS plus each said jump table;

Upon starting said native application in said host OS, said startup code goes through each said jump instruction in each said jump tables to fix the said jump instruction to point to the real address in the current running address space of said host OS;

Said startup code executing said image of guest OS in place;

~~— A method of packaging guest OS image inside a host OS application allowing in-place execution of the guest OS code to reduce memory usage where the host OS can only recognized some special file formats:~~

~~— When compiler supports, compiling guest application code to skip those areas required by host OS file format such as record headers as if they are unreachable blocks; or~~

~~Use dynamic linking and jump tables for each code segment embedded in the wrapper host application and then filling out the jump tables at run time after loading.~~

17. (Currently Amended) The method of claim 16, wherein said native application in said host OS is a database file with multiple data records or Palm PDB file with multiple Palm DB records.

~~The method of claim 16, wherein the host OS file format is a database file with multiple records or Palm PDB file format.~~

18. (Currently Amended) The method of claim 16, wherein said image of guest OS code requires one sequential memory address or one code segment, said native application of host OS contains multiple chunks whose maximum size is limited by a chunk size boundary; and Compiling a guest OS image into multiple code segments further comprising the steps of:

Compiling said image of guest OS into one code segment with sequential memory address;

Re-arrange instructions in said image of guest OS to reserve spaces to be used for jump tables on every said chunk size boundary by inserting jump instructions to by pass those spaces;

Splitting said one code segment into multiple code segments on said chunk size boundaries;

~~The method of claim 16, wherein the guest OS code requires sequential arrangement in memory.~~

19. (Currently Amended) The method of claim 16, wherein the size of a guest OS image exceeds the maximum size limit of said native application of said host OS, further comprising the steps of:

Splitting said image of guest OS into multiple pieces;